Date=3/11/2020
Lecture By=Manish Mahant
Subject ⇒ Es Linters In Javascript

IN PREVIOUS LECTURE **(QUICK RECAP) Date-2/11/2020**	**In Today's Lecture (Overview)**
JavaScript Module Bundlers What Is Webpack Javascript Transpilers Babel Transpiler How Babel Transpiler Works Task Runners In Javascript Gulp Task Runner Introduction What is gulp? How to set up gulp Node Creating a new project Questions For Self Practice / CC/Assignment For The Day	What Is Javascript linters JSLint Example Getting Started with ESLint Installation and Usage Configuration ECMAScript 2015 - ES6 New Features in ES6 JavaScript let JavaScript const Arrow Functions Question For Self-Practice // CC For The Day

What Is Javascript linters

JavaScript linters are tools that you can use to help you debug your code. They scan your scripts for common issues and errors, and give you back a report with line numbers that you can use to fix things. In addition to actual bugs and errors, they also check for subjective, stylistic preferences as well.

JSLint Example

Let's run this code through JSLint and see what problems it can detect.

```
function sayHello(name) {
    alert("Hello " + name);
}

name = "Douglas Crockford";
sayHello(name)
```

JSLint analyzes the code and reports a list of potential code quality concerns.

```
  1) Expected one space between ')'
and '{'.
function sayHello(name) }

  2) Expected 'use strict'
before 'alert'.
alert("Hello " + name);

  3) Undeclared 'name'.
name = "Douglas Crockford";

  4) Undeclared 'name'.
sayHello(name)

  5) Expected ';'
and instead saw '(end)'.
sayHello(name)
```

Getting Started with ESLint

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs. In many ways, it is similar to JSLint and JSHint with a few exceptions:

- ESLint uses Espree for JavaScript parsing.
- ESLint uses an AST to evaluate patterns in code.
- ESLint is completely pluggable, every single rule is a plugin and you can add more at runtime.

Installation and Usage

Prerequisites: Node.js (`^10.12.0`, or `>=12.0.0`) built with SSL support. (If you are using an official Node.js distribution, SSL is always built in.)

You can install ESLint using npm or yarn:

```
npm install eslint --save-dev

# or

yarn add eslint --dev
```

You should then set up a configuration file, and the easiest way to do that is to use the `--init` flag:

```
$ npx eslint --init

# or

$ yarn run eslint --init
```

Note: `--init` assumes you have a `package.json` file already. If you don't, make sure to run `npm init` or `yarn init` beforehand.

After that, you can run ESLint on any file or directory like this:

```
$ npx eslint yourfile.js

# or

$ yarn run eslint yourfile.js
```

It is also possible to install ESLint globally rather than locally (using `npm install eslint --global`). However, this is not recommended, and any plugins or shareable configs that you use must be installed locally in either case.

Configuration

Note: If you are coming from a version before 1.0.0 please see the migration guide.

After running `eslint --init`, you'll have a `.eslintrc.{js,yml,json}` file in your directory. In it, you'll see some rules configured like this:

```
{
  "rules": {
      "semi": ["error", "always"],
      "quotes": ["error", "double"]
  }
}
```

The names `"semi"` and `"quotes"` are the names of rules in ESLint. The first value is the error level of the rule and can be one of these values:

- `"off"` or `0` - turn the rule off
- `"warn"` or `1` - turn the rule on as a warning (doesn't affect exit code)
- `"error"` or `2` - turn the rule on as an error (exit code will be 1)

The three error levels allow you fine-grained control over how ESLint applies rules (for more configuration options and details, see the configuration docs).

Your `.eslintrc.{js,yml,json}` configuration file will also include the line:

```
{
  "extends": "eslint:recommended"
}
```

ECMAScript 2015 - ES6

ECMAScript 6, also known as ES6 and ECMAScript 2015, was the second major revision to JavaScript.

New Features in ES6

- The `let` keyword
- The `const` keyword
- JavaScript Arrow Functions
- JavaScript Class
- JavaScript Promise
- JavaScript Symbol
- Default Parameter Values
- Function Rest Parameter
- `Array.find()`
- `Array.findIndex()`
- New Number Properties
- New Number Methods
- New Global Methods

JavaScript let

The `let` keyword allows you to declare a variable with block scope.

Example

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

JavaScript const

The `const` keyword allows you to declare a constant (a JavaScript variable with a constant value).

Constants are similar to let variables, except that the value cannot be changed.

Example

```
var x = 10;

// Here x is 10

{

  const x = 2;

  // Here x is 2

}

// Here x is 10
```

Arrow Functions

Arrow functions allows a short syntax for writing function expressions.

You don't need the `function` keyword, the `return` keyword, and the curly brackets.

```
// ES5

var x = function(x, y) {

  return x * y;

}

// ES6
```

```
const x = (x, y) => x * y;
```

Arrow functions do not have their own `this`. They are not well suited for defining object methods.

Arrow functions are not hoisted. They must be defined before they are used.

Using `const` is safer than using `var`, because a function expression is always a constant value.

You can only omit the `return` keyword and the curly brackets if the function is a single statement. Because of this, it might be a good habit to always keep them:

Example

```
const x = (x, y) => { return x * y };
```

Based on	ES5	ES6
Definition	**ES5 is the fifth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International)**	**ES6 is the sixth edition of the ECMAScript (a trademarked scripting language specification defined by ECMA International).**
Release	**It was introduced in 2009.**	**It was introduced in 2015.**
Data-types	**ES5 supports primitive data types that are string, number, boolean, null, and undefined.**	**In ES6, there are some additions to JavaScript data types. It introduced a new primitive data type 'symbol' for**

		supporting unique values.
Defining Variables	**In ES5, we could only define the variables by using the var keyword.**	**In ES6, there are two new ways to define variables that are let and const.**
Performance	**As ES5 is prior to ES6, there is a non-presence of some features, so it has a lower performance than ES6.**	**Because of new features and the shorthand storage implementation ES6 has a higher performance than ES5.**
Support	**A wide range of communities supports it.**	**It also has a lot of community support, but it is lesser than ES5.**
Object Manipulation	**ES5 is time-consuming than ES6.**	**Due to destructuring and speed operators, object manipulation can be processed more smoothly in ES6.**
Arrow Functions	**In ES5, both function and return keywords are used to define a function.**	**An arrow function is a new feature introduced in ES6 by which we don't**

		require the function keyword to define the function.
Loops	**In ES5, there is a use of for loop to iterate over elements.**	**ES6 introduced the concept of for...of loop to perform an iteration over the values of the iterable objects.**

Question For Self-Practice // CC For The Day

https://leetcode.com/problems/valid-palindrome/